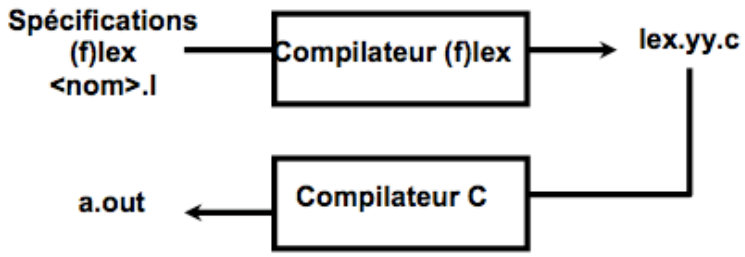
Chapitre 3

OUTIL (F)LEX

1. INTRODUCTION

- (F)LEX accepte en entrée des spécifications d'ULs sous forme de DRs et produit un programme (un analyseur lexical) écrit dans un langage de haut niveau (ici le C) qui, une fois compilé, reconnaît ces ULs.



- Le fichier de spécifications (f)lex contient des ERs suivies d'actions (règles de traduction).
- L'exécutable obtenu lit le texte d'entrée caractère par caractère jusqu'à ce qu'il trouve le plus long préfixe qui corresponde à l'une des ERs.
- Dans le cas où plusieurs règles sont possibles, c'est la 1ère règle rencontrée (de haut en bas) qui l'emporte. Il exécute alors l'action correspondante.
- Dans le cas où aucune règle ne peut être sélectionnée, l'action par défaut consiste à copier le caractère lu en entrée vers la sortie (l'écran)

2. STRUCTURE DU FICHIER DE SPÉCIFICATION (F)LEX

```
%{
déclaration (en C) de variables, constantes,…
%}
déclarations de définitions régulières
%%
règles de traduction
%%
bloc principal et fonctions auxiliaires
```

- Une DR permet d'associer un nom à une ER (f)lex et de se référer par la suite (dans la section des règles) à ce nom plutôt qu'à l'ER.
- Les règles de traduction sont des suites d'instructions de la forme :

exp_1	$action_1$
exp_2	$action_2$
…	…

- Les exp_i sont des ERs (f)lex qui doivent commencer en colonne 0.
- Les $action_i$ sont des blocs d'instructions en C

- La section bloc principal et fonctions auxiliaires est facultative (ainsi que la ligne %% qui la précède). Elle contient les routines C définies par l'utilisateur et une fonction main() si celle par défaut ne convient pas.

3. EXPRESSIONS RÉGULIÈRES (F)LEX

- Une ER (f)lex se compose de caractères normaux et de méta-caractères qui ont une signification spéciale **$, \, ^, [,], {,}, (,), +, -, *, /, |,?**

C	Tout caractère c qui n'est pas un méta- caractère	a
\c	Si ***c*** est un méta-caractère alors le caractère c littéralement	\+, \.
« s »	La chaîne de caractère ***s*** littéralement	« abc *+ »
r1|r2	***r1*** suivie de ***r2***	Ab
.	N'importe quel caractère, excepté le cacractère « à la ligne »	a.b
^	Comme premier caractère de l'expression, signifie le début de ligne	^abc
$	Comme dernier caractère de l'expression, signifie la fin de ligne	abc$
[s]	N'importe lequel des caractères constituant la chaîne ***s***	[abc]
[^s]	N'importe lequel des caractères	[^abc]
r*	0 ou plusieurs occurrences de ***r***	a*
r+	1 ou plusieurs occurrences de ***r***	a+
r?	0 ou 1 occurrence de ***r***	a?
r{m}	***m*** occurrences de ***r***	a{3}
r{m,n}	***m*** à ***n*** occurrences de ***r***	a{5,8}

r1|r2	***r1*** ou ***r2***	a|b
r1/r2	***r1*** si elle est suivie de ***r2***	ab/cd
(r)	***R***	(a|b)?c
\n	Caractère « à la ligne »	
\t	Tabulation	
{}	Pour faire référence à une DR	{NOMBRE}
EOF	Fin de fichier (uniquement avec flex)	

ATTENTION

- (f)lex fait la différence entre les minuscules et majuscules
- **$**, ^ et / ne peuvent apparaître dans des () ni dans des DRs
- ^ perd sa signification *début de ligne* s'il n'est pas au début de l'ER
- **$** perd sa signification *fin de ligne* s'il n'est pas à la fin de l'ER
- à l'intérieur des **[ ]**, seul \ reste un méta-caractère, le – ne le reste que s'il n'est ni au début ni à la fin de l'ER.

PRIORITÉS

- a|b* est interprété comme (a)|(b*)
- abc{1,3} est interprété avec lex comme (abc){1,3} et avec flex comme ab(c{1,3})
- ^a|b est interprété avec lex comme (^a)|b et avec flex comme ^(a|b)

4. VARIABLES ET FONCTIONS PRÉDÉFINIES

- **char yytext[ ]** : tableau de caractères qui contient la chaîne qui a été acceptée,
- **int yyleng** : longueur de cette chaîne,
- **int yylineno** : numéro de la ligne courante (lex),
- **yyin** : fichier de lecture (par défaut : stdin)
- **yyout** : fichier d'écriture (par défaut stdout)
- **int yylex()** : fonction qui lance l'analyseur (et appelle **yywrap()**),
- **int yywrap()** : fonction toujours appelée en fin de flot d'entrée. Elle retourne 0 si l'analyse doit se poursuivre (sur un autre fichier d'entrée) et 1 sinon. On peut la redéfinir dans la section des fonctions auxiliaires.
- **yyterminate()** : fonction qui stoppe l'analyseur (flex).

5. EXEMPLES DE FICHIER.L

Exemple 1 : reconnaît les nombres binaires :

```
%%
(0|1)+          printf ("un nombre binaire") ;
```

NB : tout ce qui n'est pas reconnu comme nombre binaire est affiché à l'écran

Exemple 2 : Autre version qui n'affiche que les nombres binaires reconnus :

```
%%
(0|1)+          printf ("un nombre binaire %s",yytext) ;
.               // rien
```

Exemple 3 : supprimer les lignes qui commencent par *p* ainsi que tous les entiers, remplacer les *o* par des * et retourner le reste inchangé :

```
chiffre         [0-9]
entier          {chiffre}+
%%
{entier}        printf ("j'élimine les entiers") ;
^p(.)*\n        printf ("j'élimine les lignes qui commencent par p") ;
[oO]            printf ("*") ;
.               printf ("%c",yytext[0]); //action faite par défaut
```

Exemple 4 : compter le nombre de voyelles, consonnes et caractères de ponctuations d'un texte entré au clavier :

```
%{
int nbVoyelles, nbConsonnes, nbPonct ; // déclarations des variables
%}
consonne                [b-df-hj-np-xz]
ponctuation             [, ;:\?!\.]
%%
[aeiouy]                nbVoyelles++ ;
{consonne}              nbConsonnes++ ;
{ponctuation}           nbPonct++ ;
.|\n                    //ne rien faire
%%
int yywrap ( ) {return 1 ;}

main( )
{       nbVoyelles=nbConsonnes=nbPonct=0 ;
        yylex( ) ;
        printf ("il y a %d voyelles, %d consonnes et %d ponctuations.\n", nbVoyelles,
        nbConsonnes, nbPonct) ;
}
```

Exemple 5 :

```
%{
int  nb;
%}

MAJ  [A-Z]

%%
{MAJ}+  { printf ("[%s] : uniquement des majuscules",yytext);   nb++ ; }
.*          { printf ("[%s] : pas uniquement des majuscules ",  yytext); }
%%

int yywrap ( ) {return 1;}

main ( )
{
        nb=0;
        yylex( );
        printf ("Le nombre des lignes contenant uniquement des majuscules est %d",nb) ;
}
```